PCA

- Reduce dimensionality of large data sets ->smaller sets contain most of the initial info
- < #variables, trade accuracy for simplicity
- Smaller sets: easier and faster to explore, analyse and visualize
- Goal: reduce variables in set while preserving as much info as possible
- Very sensitive regarding variances of initial variables -> need to do standardization
- In more detail:

  1st step – Standardize the range of continuous initial variables so that all have the same contribution to the analysis. Is important since PCA is sensitive to variances of the initial variables (the ones with larger ranges will dominate over the smaller ones, leading to biased results)

  $$z = \frac{value - mean}{standard\ deviation}$$

  2nd step – Compute Covariance Matrix: check for relationship(correlation) between the variables of the input data set. Cov matrix is pxp (p dimensions) and symmetric. If positive: variables are correlated (both increase both decrease). If negative: one increase when the other decreases, meaning they are inversely correlated.

  3rd step – Compute eigenvalues and eigenvectors: From the Cov matrix, both give the principal components of the data (new variables constructed as a linear combination/mix of the initial variables, leading to p components for a p-dimensional data). The first component gives the most information and accounts for the largest possible variance (more information) – the Principal Component –, the second gives the maximum information remaining after the first component, and so on, this way not much information is lost with reducing dimensionality. Regarding geometry, principal components represent the directions of data that explain maximal amount of variance of the data.

  4th step – Feature Vector – a matrix where its columns are the eigenvector of the components chosen in the previous step

  5th step – Recast the data along the principal component axes (reorient the data form the original axes to ones represented by the principal components).

  $$FinalDataSet = FeatureVector^T * StandardizedOriginalDataSet^T$$

RMSprop Optimizer – uses an adaptive learning rate instead of treating the learning rate as a hyperparameter, so this way the learning rate changes over time. Results: higher accuracy for large epochs compared to static learning rate and Adam optimizer. Impacts the NN such that it converges to the final result faster, but does not change the result if, for example, the model is overfitted.

Adam optimizer – Another first-order-gradient-based algorithm of stochastic objective functions. Its update rule is based on the first moment normalized by the second moment gives the direction of the update.

$$\theta_{n+1} = \theta_n - \frac{\alpha}{\sqrt{\hat{v}_n} + \epsilon}\hat{m}_n$$

Multi-tasking – Sharing representations between related tasks, enabeling the model to generalize better on the original task. This can happen, for example, if more than one loss function is being optimized for a model. Hard parameter sharing greatly reduces the risk of overfitting. In soft parameter sharing on the other hand, each task has its own model with its own parameters. The distance between the parameters of the model is then regularized in order to encourage the parameters to be similar.

End-to-end Learning – Enable to train a complex learning system by applying grandient-based learning to the system as a wholde. Allows to scale up the backpropagation algortihm to more rich and complex models. Ensure that all modules of a learning system are differentiable with respect to all adjustale parameters (weights) and trainning this system as a whole.

Effect of XX in NN

- Batch size – larger batch size allows computational speedups from the parallelism of GPUs. If it's too large the batch size will lead to poor generalization. Small batches show to have a faster convergence to "good" solutions and allow the model to "start learning before having to see all the data". However, very small batches may lead the model to not converge to the global optima.
- Batch normalization - After using batch normalization, the new model is likely to have a smaller gap between the train and validation accuracies, if before the trainning accuracy was significantly higher than the one verified in the validation set.
- Learning Rate – Deep learning networks are trained using the stochastic gradient descent optimization algorithm. Learning rate usually is between 0.0 and 1.0. Too-large learning rate show oscialtions in behaviour. With too-small values, inability of the model to learn anything is verified.
- Bias – Helps in controlling the value at which the activation function is triggered.

Variance of a model - The difference between validation error and training error. Adding more data, adding regularization/dropout, creating a smaller model, etc are possible ways to reduce a high variance scenario.

Non Linearity – Introduces degrees of freedom to the model.

L1 vs L2 regularization

- For small weights, L1 regularization will push them to zeros while L2 will not. L1 regularization can lead to sparse weights.
- For large weights, L1 shifts them towards zero by a fixed distance, while L2 will shift larger weights by longer distances.

The Bias-Variance tradeoff - More complex models like deep neural networks have low bias but suffer from high variance. This implies that, these models overfit the training data and would show poor perform on test data or the data, they haven't seen before. This would lead to *higher prediction errors*. Thus, the increased diversity from data augmentation reduces the variance of the model by making it better at generalizing.

Higher bias -> lower variance : overfitting

Lower bias -> higher variance : underfitting (how to solve: add layers/learnable paramenters)

Extra: More data -> lower variance
Weight Sharing -> higher bias (?)

Data augmentation may lead to underfit - In contrast to overfitting, your model may be underfitting because the training data is too simple. It may lack the features that will make the model detect the relevant patterns to make accurate predictions. Adding features and complexity to your data can help overcome underfitting.

Vanishing Gradient Solution – (recap: gradients are too small to progress, most frequently for large values and sigmoid activation function). Solution: Use other activation functions, such as ReLU or Imprement residual networks, which provides residual connections straight from ealier layers – that don't go through activation functions that "squashes" the derivatives – or batch normalization – which normalizes the input so |x| doesn't reach the outher edges of sigmoid fucntion, leading to a very small derivative or use Long Short-Term Memory Networks (LSTMN) in case of a recurrent nn – transform the rnn into a deep multilayer perceptron.

Exploding Gradient Solution – (recap: gradients acumulate during training and if the results are too large, the weights will be updated with very large variations, leading to an unstable network). Solution: re-design network (to have fewer layers, use a smaller batch size in trainning and in recurrent nn, update across fewer prior time steps during training (truncated Backpropagation through time)) or use weight regularization – apply a penalty to the networks loss function for large weight values nn – or use gradient clipping – check for and limit the size of gradients during the training of the network, if the values of the error gradient exceeds a threshold are clipped or set to that threshold value.

Cross-Entropy – Calculates the difference between two probability distributions usually.

Autoenconders – NN that can be trained for the task of representation learning. Attempt to copy its input to its output through a combination of an encoder and a decoder. Usually, autoencoders are trained using recirculation, a learning algorithm based on comparing the activation of the network of the input to the activation of the reconstructed input. Traditionally used for the task of dimensionality reduction or feature learning. The fundamental problem with autoencoders is that the latent space they convert their inputs to and where their encoded vectors lie, may not be continuous, or allow easy interpolation.

Under-complete autoencoder – Are autoencoders whose dimensions are less than the input dimension. By penalizing the network according to the reconstruction error, the model can learn and capture the most salient features. Uses the entire network for every observation.

Regularised autoencoder – As under-complete autoencoders, fail to learn anything if the bottleneck has the same, or higher, dimension as the input. They use a loss function that enables this model to learn useful features, such as sparcity and smallness of the derivative of the representation as well as robustness to noise or to missing inputs.

Sparce autoencoder – Type of model that has been regularised to respond to unique statistical features. Is forced to selectively activate regions of the network depending on the input data, eliminating the network capacity to memorise the features from the input data and enables it to learn useful information and features. Sparcity constrain imposed by L1 regulariazation or KL-divergence, where both involve measuring the hidden layer activations for each training batch and adding a term to the loss function to penalize excessive activations.

Denoising autoencoder – Model learns a vector fiel for mapping the unut data torwards a lower-dimensional manifold, instead of developing a mapping strategy based on traning data memorization. This manifold accurately describes the natural, non-corrupted data.

Variational autoencoder – A generative model which generates continuous latent variables that use learned approximate inference. Different from vanilla enconders since its latent space is, by design, continuous, allowing easy random sampling and interpolation. This is achieved by making its encoder not having a random latent variable as an output, but two vectors: vector of

means and a vector of standart deviations, that will follow a normal distribution. Achieves what idealy is ask out of encodings, which is all of them being as close as possible to each other while still being distinct, allowing smooth interpolation and enabling the construction of new samples. Its drawback is that the samples generated from the model might be blurry sometimes.

Generative Models – Unsupervised learning task that involves automatically discovering and learning the representations or patterns in input data such that the model can be used to generate new examples. All the models represent probability distributions over multiple variables in some manner. The distributions that the generative model generates are high-dimensional.

Boltzmann Machines – Is a NN of symmetrically connected nodes that make their own decisions (to activate or stay idle). They use a straightforwards stochastic leaning algorithm to find features that represent complex patterns in the input data and, architecture-wise, only have visible (input) and hidden nodes, so no output. The inter-connected nodes as called states. The inputs are connected and exchange information among themselves and self-generate subsequent data, allowing them all to learn all the parameters, their patterns and correlation amoung the data. It is generally quite slow in networks with xtensive feature detection layers and fast on a single layer of feature detection (restricted boltzmann machines).

Variational Inference – Model that consists of two important components: (1) tractable components, wich are differentiable and continuous making the model deterministic; (2) intractable components, which make the model flexible and probabilistic.

KL Divergence – measure og the difference between two distributions, how much information we lose when we choose an approximation.

The Information Bottleneck – Extracting relevant information by compressing the amount of information that can traverse the full network, forces a learned compression of the input data, that has a reduced dimension but also reduced complexity of the data. The ideia is that by shrinking the information through a bottleneck would remove the noise in input data leaving the most relevant features which are the most relevant to general concepts.

Latent variables – Is a random variable that cannot be observed directly but it lays the foundation of how the data is distributed, giving us a low-level representation of high-dimensional data, an abstract representation of the distribution of the data. Models with latent variables can be used to perform a generative process, the so called generative models.

Parameter Sharing – CNN has been able to reduce the unique model parameters quite significantly, while being able to increase the model size at the same time. This sharing of parameters causes the layers to have a property called Equivariant representations, which means that if the input changes, the output changes in the same way. This way we can prevent the model from overparameterization and overfitting.

GANs – Based on a competition of two networks: generator and discriminator. The generator generates samples and the discriminator is a function that distinguishes smaples form the real dataset and the generator. During trainning, the generator tries to be better at generating real-looking images, while the discriminator trains to better classify those images as fake (a normal classification model). An equilibrium is reached when the discriminator can no longer distinguish real images from fake ones. After training, the generator point its multidimensional vector space will correspond to points in the readl data, forming a compressed representation of the data distribution as latent variables, while the discriminator model is discarted. This model discovers

underlying features and then exploits them in different ways to produce new samples of images. GAN is a differential model, which means that the parameters in the generator model can update itself using gradient descent. Meanwhile, a discriminator model maximizes the loss function using gradient ascent.

Embedding – The process of learning another set of vector values from the input data, and expressing the original actual vector through anoter set of vectors. In the context of nn, embeddings are low-dimensional, learned continuous vector representations of discrete variables. NN embeddings are useful since they can reduce the dimensionality of categorical variables and meaningfully represent categories in the transformed space.

Word Embedding – Type o word representation that allows words with similar meaning to have a similar representation. Each word is mapped to one vector and the vector alues are learned in a way that resembles a NN. The key is to use a dense distributed representation for each word, so that we have a tens or hundreds of dimension real-valued vector, instead of thousands or milions of dimensions for sparese word representation, as one-hot encoding.

Learn an Embedding – requires a large amount of text data to ensure useful embeddings are learned.

Learn it Standalone – model trained to learn the embedding, which is saved and used as a part of another model for the task later. Same embedding -> multiple models

Learn Jointly – embedding learned as part of a large task-specific model. One embedding -> one task.

Reuse an Embedding – Pre-trained word embeddings that can be used on a project instead of training from scratch your own embeddings. Pre-trained embeddings:

Static – kept static and used as a component of the model

Updated – Pre-trained embedding used to seed the model, but is updated jointly during the training of the model.

Matrix Factorization – An embedding technique used to remove similar feature vectors. Does a good job at using global text statistics but not as good as learned methods like word2vec at capturing meaning and demonstrating it on tasks like calculationg analogies.

Word2vec – An embedding technique widely used in natural language processing, that allows efficient learning a standalone word embedding from a text corpus. Encodes words into vectors. The input vector is fed into the neural nets, then it uses those vectors to characterize the semantic formation of words by learning text trough an embedding space (latent space) that makes semantically similar words very cole in this space. Has two main models: Skip-Gram – given the input work, predicts the context – and Continuos Bag of Words (CBOW) – predicy the input word given the context.

GloVe – The Global Vectors for Word Representation algorithm is an extenson to the word2vec method. Combines the global statistics of matrix factorization techniques like LSA (classical vector space model representation of words) with the local context-based learning in word2vec. Instead of defining local context, it constructs an explicit word-context or word co-occurrence matrix using statistics across the whole text corpus. The result is a learning model that mau resul in generally better word embeddings.

Pre-training - Training the machines, before they start doing particular tasks. The first network is your pre-trained network. The second one is the network you are fine-tuning. Random initialisation of inputs is random, the values of the weights have nothing to do with the task your trying to solve. Therefore, Pre-training gives the network a head start. As if it has seen the data before. A pre-trained model can be used in two ways:

As a feature extractor: Train model on a bigger data (similar domain for example), use on the required problem as on "off-the-shelf" model

As a fined-tuned-network: Train model on a bigger data, the freeze the upper layers (set learning rate to minimum in early layers) and retrain on the required problem.

Fine-tuning - Not only replace FC layers of a pretrained model with a new set of FC layers to retrain them on a given dataset, but to fine-tune all or part of the kernels in the pretrained convolutional base by means of backpropagation. FC, fully connected. All the layers in the convolutional base can be fine-tuned or, alternatively, some earlier layers can be fixed while fine-tuning the rest of the deeper layers.

One drawback of transfer learning is its constraints on input dimensions. The input image must be 2D with three channels relevant to RGB because the ImageNet dataset consists of 2D color images that have three channels (RGB: red, green, and blue), whereas medical grayscale images have only one channel (levels of gray).

Adversarial Attacks - Attacks against AI models are often categorized along three primary axes — influence on the classifier, the security violation, and their specificity — and can be further subcategorized as "white box" or "black box." In white box attacks, the attacker has access to the model's parameters, while in black box attacks, the attacker has no access to these parameters.

An attack can influence the classifier — i.e., the model — by disrupting the model as it makes predictions, while a security violation involves supplying malicious data that gets classified as legitimate. A targeted attack attempts to allow a specific intrusion or disruption, or alternatively to create general mayhem.

Evasion attacks are the most prevalent type of attack, where data are modified to evade detection or to be classified as legitimate.

Encoder-decoder RNNs/LSTMs

We have two RNNs/LSTMs: one encoder (reads the input sentence and makes sense out of it, summarizes it and passes the summary – context vector – to the decoder) and one decoder that translates the input sentecen by just seeing it.

- The encoder LSTM is used to process the entire input sentence and encode it into a context vector, which is the last hidden state of the LSTM/RNN. This is expected to be a good summary of the input sentence. All the intermediate states of the encoder are ignored, and the final state id supposed to be the initial hidden state of the decoder
- The decoder LSTM or RNN units produce the words in a sentence one after another
- Drawback: If the encoder fails in summarizing correctly the translation will be wrong, which is verified for example longer sentences (the long-range dependency problem of RNN/LSTMs) and it's not possible to give more importance to some of the input words compared to others while translating the sentence.

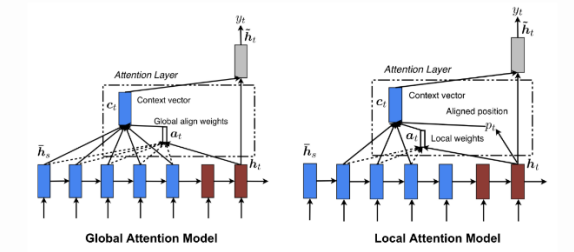
Long-range dependency problem of RNN/LSTMs – RNNs cannot remember longer sentences and sequences due to the vanishing/Explofind gradient provlem. It can remember the parts which it has just seen. (LSTMs are supposed to capture this long-range dependency better than RNN)

Possible solution: Context vector taking into account not only the input words but also relativa importance associated to each of them. This way, when generating a sequence the model searches for a set of positions in the encoder hidden states where the most relevant information is available: Attention.

Attention

Global – All the hidden states are taken into consideration, concatenated into a matrix and multiplied with a mtrix of correct dimensions to get the final layer of the feedforwards connection, involving therefore a lot of computation. If the input size increases, so does the matrix size, meaning that an increase of the number of feedforwards connection leads to an increase of computation. How to reduce this? Use Local attention.

Local – Instead of considering all the encoded inputs, only a part is considered for the context vector generation, avoiding expensive computation.



Self-Attention – Mechanism of relating different positions of a single sequence or sentence in order to gain a more vivid representation.

Each embeeding of a word should have three different vectors corresponding to it: Key, Query and Value. They are abstractions of the embedding vectors in different subspaces. So, to calculate the attention of a target word with respect to the input embeddings, you should use the Query of the target and the key of the input to calculate a matching score, and these matching scores then act as the weights of the Value vector during summation.
(tip: you raise a query. The query hits the key of the input vector. The key can be compared with the memory location you read from, and the value is the value to be read from the memory location).

If there are 3-headed self-Attention, corresponding to a word, there will be 3 different Z matrices also called “Attention Heads”. These Attention heads are concatenated and multiplied with a single weight matrix to get a single Attention head that will capture the information from all the Attention heads.

Applications:

Image Captioning - In image captioning, a convolutional neural network is used to extract feature vectors known as annotation vectors from the image. This produces L number of D dimensional feature vectors, each of which is a representation corresponding to a part of an image.

Image Generation – Draw parts of an image sequentially, just like an artist would, using a variational autoencoder dedicated to image generation (generates N guassian distributions, form which N elemente latent vector is samples and after fed to the decoder for the output image generation)

Softmax function: $$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)},$$

Quick notes and solutions:

Underfit – increase capacity of the network

Overfit – regularization (L1, L2, dropout (increase prob p)), early stopping, simplifying model, data augmentation(flips, brightness, scale – but do not create artificial images!)

Small dataset – K-fold cross validation, data augmentation, transfer learning